Journal homepage http://revistas.unitru.edu.pe/index.php/SSMM

SELECCIONES MATEMÁTICAS
Universidad Nacional de Trujillo
ISSN: 2411-1783 (Online)
2023; Vol. 10(1): 199-209.

REVIEW

Brief review of classical Effort Estimation models for Software development projects

Breve revisión de los modelos clásicos de Estimación de Esfuerzo para proyectos de desarrollo de Software

Diego Bravo-Estrada and Roxana López-Cruz

Received, Jan. 21, 2023; *Accepted, Apr. 30, 2023;* *Published, Jul. 27, 2023*

How to cite this article:
Bravo-Estrada D, López-Cruz R. *Brief review of classical Effort Estimation models for Software development projects*. Selecciones Matemáticas. 2023;10(1):199–209. http://dx.doi.org/10.17268/sel.mat.2023.01.17

Abstract

A critical synthesis on the most representative models for software development project effort estimation is provided. This work is a basis for a discussion about the methodological and practical challenges which entail the effort estimation field, specially in the mathematical/statistical modelling fundamentals, and its empirical verification in the software industry.

Keywords . Software effort estimation, software planning, software engineering.

Resumen

Se proporciona una síntesis crítica de los modelos más representativos propuestos en la literatura para la estimación de esfuerzo en proyectos de desarrollo de software. Este trabajo sirve de base para una discusión acerca de las dificultades metodológicas y prácticas que enfrenta el campo de la estimación de esfuerzo, especialmente en la fundamentación de los modelos matemático/estadísticos más populares, así como su verificación empírica en la industria del software.

Palabras clave. Estimación de esfuerzo en software, planificación de software, ingeniería de software.

1. Introducción. Desde el nacimiento de la "ingeniería de software" a fines de los sesenta [24], el problema de la planificación de los proyectos ha sido una de las preocupaciones más importantes de esta disciplina; así, no es de extrañar que a lo largo de los años haya sido abordado desde diversos ángulos en numerosas publicaciones; en particular, en lo referido a la estimación de los recursos humanos necesarios, los plazos de ejecución, y los costos derivados.

En los últimos años la planificación de proyectos (de toda clase) se ha encuadrado en lo que podemos denominar la "subdisciplina" de gestión de proyectos[1]; ésta integra los requerimientos e informaciones provenientes de diversos especialistas y actores relevantes, a fin de materializar un plan de trabajo satisfactorio y gestionable.

En la planificación de proyectos de desarrollo de software, es usual (y razonable) considerar la "opinión" de los desarrolladores que estarán a cargo de aquéllos; esta opinión puede ser aquilatada de modo informal, o ulteriormente procesada mediante la aplicación de una amplia gama de métodos formales, los que suelen estar basados en algún modelo matemático y/o estadístico.

Cuando las estimaciones son elaboradas esencialmente a partir de las opiniones (informadas pero necesariamente subjetivas) de los especialistas, se dice que el método empleado ha sido "basado en el experto"

*Unidad de Posgrado, Facultad de Ciencias Matemáticas, UNMSM, Lima, Perú (diego.bravol@unmsm.edu.pe).
†Departamento de Matemáticas, Universidad Nacional Mayor de San Marcos, Lima, Perú (rlopezc@unmsm.edu.pe).
[1]La referencia usual es [41]. Un enfoque alternativo se puede consultar en los primeros capítulos de [46].

(expert-based.) En líneas generales, estos métodos intentan consolidar la opinión de un conjunto de profesionales mediante cierto procedimiento preestablecido, a fin de evitar una serie de inconvenientes de orden interpersonal y/o psicológico; por ejemplo, la influencia exagerada de un experto de mayor jerarquía en la organización, el exceso de optimismo de los directivos, etc.

Los métodos basados en el experto han sido y son los más empleados a juzgar por las investigaciones realizadas [37, 2].

La alternativa a éstos corresponde al uso de "herramientas de estimación", las cuales se emplean como complemento a las herramientas generales de planificación de proyectos; según [19], el uso de estas herramientas de estimación es un factor clave para el éxito de proyectos de gran extensión. Las herramientas de estimación se basan en diversos métodos para obtener sus resultados, pero éstos típicamente se concentran alrededor de un modelo matemático y/o estadístico. Una sub-especialización muy activa en los últimos años corresponde a modelos basados en técnicas de inteligencia artificial, los cuales no serán considerados en el presente estudio[2].

Los métodos y modelos de estimación de esfuerzo son muy variados y se clasifican de diversas formas según los autores que se consulte[3]. Por ejemplo, según [49] éstos se clasifican en

a) "orientados a los datos" (data-driven) y

b) basados en la opinión del experto.

Los modelos "orientados a los datos" se subclasifican en "propietarios" y "no propietarios"; éstos últimos a su vez se subdividen entre los que están "basados en modelos" y aquellos que son "basados en analogía". En todos los casos se puede definir categorías "híbridas" que combinan las ya mencionadas[4]

En relación a los principales métodos utilizados en la actualidad, existen diversas síntesis que dan cuenta del "estado del arte", por ejemplo [39, 52, 28][5]

Un rasgo inherente a todas las ingenierías es el uso de métodos y modelos con base científica (léase, objetivos) para el análisis de sus problemas; en el caso de la estimación del esfuerzo del software, es natural cuestionar por qué mayoritariamente se prefiere la "opinión del experto", abriéndose la puerta a la subjetividad.

Al respecto, en la literatura se puden hallar diversas respuestas; sin embargo, a nuestro entender la explicación más pragmática consiste en aceptar que a la fecha los métodos formales de estimación de esfuerzo no han probado ser superiores en capacidad predictiva a la opinión del experto. Pese a esto, los métodos formales se siguen empleando y recomendando en la industria, especialmente para proyectos de gran magnitud [25, 36].

En la siguiente sección se pasa revista a una selección de antecedentes basados en modelos matemáticos y estadísticos que han tenido gran relevancia en la literatura académica, los cuales servirán como materia prima para la discusión crítica subsecuente. Las consecuencias de esta argumentación se exponen en las conclusiones.

2. Revisión de Antecedentes.

A continuación presentamos una revisión de algunos antecedentes relacionados con los modelos de estimación de esfuerzo de software que consideramos importantes e ilustrativos. Las referencias anteriormente citadas contienen estudios mucho más amplios y detallados.

2.1. Software Science.

A lo largo de la década de los setenta, Halstead [14] elaboró un conjunto de indicadores con la finalidad de cuantificar diversas características intrínsecas al texto de los programas (listados de código fuente.) Estas métricas intentaban proporcionar información más significativa que el tradicional "conteo de líneas" (de código fuente), así como establecer relaciones con el "esfuerzo" dedicado por el desarrollador en su respectiva elaboración.

A tal efecto investigó el uso (conteo) de las unidades constituyentes del texto de los programas (palabras clave, operadores, operandos), estableciendo dos expresiones que resumirían las características cuantitativas del código fuente: estas fueron bautizadas como "volumen" y "nivel", las cuales expresan (en términos muy generales) la extensión del vocabulario utilizado y la variabilidad relativa operadores y operandos presentes en los módulos que componen los programas.

Experimentalmente observó que el producto de volumen y nivel se mantenía constante; esto unido a un (forzado) análisis del supuesto modelo mental del programador frente a una visión simplificada de lo que involucra implementar un algoritmo, le permitió proponer una fórmula para el número de "discriminaciones

[2] La literatura basada en técnicas basadas en inteligencia artificial es abundante; sin embargo, adolece de los mismos inconvenientes relacionados al uso de "datasets" que describimos más adelante. Una crítica punzante puede consultarse en [23] donde el autor se pregunta con ironía: ¿De cuántas formas se puede hacer 'machine learning' sobre datasets conteniendo menos de 100 filas?

[3] Para una discusión acerca de las taxonomías empleadas en la literatura, consultar [50].

[4] Los métodos "propietarios" se comercializan sin proporcionar al público los detalles de su implementación (salvo quizá, las ideas generales de su funcionamiento.) Los métodos "basados en analogía" consisten esencialmente en consultar proyectos de desarrollo anteriores que sean similares (análogos) a lo que se desea construir: la estimación corresponde a una extrapolación del esfuerzo que fue requerido por los anteriores desarrollos.

[5] Publicaciones más orientadas hacia la puesta en práctica de estos métodos pueden consultarse en [35, 32, 9].

mentales" necesarias para su implementación; a su vez, estas discriminaciones se identifican con el concepto de "momento psicológico" propuesto en algunas investigaciones de la psicología conductual de la época. Así pudo obtener una fórmula para el tiempo requerido (por la mente) para el desarrollo del algoritmo, en función de una variable asociada a la velocidad cerebral (ratio de "momentos" del cerebro por unidad de tiempo.)

No obstante el interés inicial recibido, esta propuesta nunca cumplió con sus metas. En [11] se puede consultar una crítica a su sospechosa aplicación de resultados de la psicología, mientras que en [45] se resumen diversas críticas inherentes a la construcción cuantitativa de la teoría, así como su constatación experimental.

2.2. Complejidad Ciclomática. Desarrollada por McCabe [34], esta métrica intentó cuantificar la complejidad de un programa o algoritmo mediante el análisis de las posibles rutas y/o ciclos de ejecución[6]. En ese sentido el programa se modela como un grafo G que representa los flujos que éste recorre a lo largo de su ejecución[7]. La complejidad está dada por el "número ciclomático":

$$V(G) = e - n + p,$$

donde n representa el número de vértices, e el número de aristas y p el número de componentes conectados.

El autor propuso diversas equivalencias de las secuencias de control típicas de los programas (notablemente en lenguaje Fortran) a fin de automatizar el cálculo de la complejidad. Esta medida se asocia con frecuencia a las tareas de control de calidad en tanto que (al menos en teoría) cada posible flujo del programa genera sendos casos de prueba a ser verificados.

Numerosos autores han propuesto adecuaciones a la complejidad ciclomática, así como diversas extensiones para ampliar su margen de aplicación (por ejemplo, existen diversas propuestas dirigidas a lenguajes orientados a objetos.) El ya mencionado [45] también proporciona críticas a la conceptualización y utilidad de la complejidad ciclomática.

A nuestro entender, estas métricas de complejidad intrínsecas al flujo de la ejecución de los programas involucran al menos dos inconvenientes de cara a su potencial en la estimación de esfuerzo:

a) requieren disponer de un diseño muy detallado acerca de los programas a ser desarrollados. Este diseño detallado (de llevarse a cabo), involucra un esfuerzo significativo en el proyecto, siendo comparable con la codificación en sí; esto quiere decir que la estimación de esfuerzo se obtendría sólo después de haberse invertido un gran porcentaje del esfuerzo total, con lo que aquella pierde gran parte de su valor predictivo.

b) en muchas aplicaciones (especialmente comerciales) el flujo de ejecución no es el factor que demanda mayor esfuerzo en el proceso de desarrollo: los algoritmos más complejos suelen estar ya codificados en componentes reutilizables (como sistemas de gestión de base de datos, librerías matemáticas, etcétera), por lo que en términos relativos, el valor de estas estimaciones sería poco significativo.

2.3. Puntos de Función. El análisis de puntos de función [1] proporciona un indicador que cuantifica la "funcionalidad" que debe brindar un programa desde la perspectiva del usuario, y se obtiene sumando diversos constituyentes ponderados por su respectiva complejidad.

Existen diversas variantes para el análisis de puntos de función promovidas por grupos de usuarios y organizaciones[8]; con fines ilustrativos exponemos a continuación las ideas principales del estándar IFPUG (International Function Point Users Group) [16]:

El cálculo de los puntos de función IFPUG correspondiente a un componente de software o programa, se inicia con el cálculo de un valor "no ajustado" de sus puntos de función (UFP = unadjusted function points):

$$UFP = (c_1{\times}I) + (c_2{\times}O) + (c_3{\times}Q) + (c_4{\times}F) + (c_5{\times}E),$$

donde I es el número de "entradas" de cara al usuario, O el número de "salidas" hacia el usuario, Q el número de procesos de interacción con el usuario, F el número de archivos principales (maestros) en el sistema, y E el número de interfaces con otros sistemas. Las constantes c_i son factores de ponderación tabuladas a partir de registros históricos de proyectos del pasado, y se proporcionan en la documentación del estándar.

A continuación se incorpora un conjunto de "factores de influencia" asociados al entorno de operación del software en cuestión, que potencialmente lo hacen más complejo; por ejemplo, exigencias de alta performance, múltiples nodos de ejecución, necesidades de reutilización para futuros proyectos, requerimientos

[6] Existen diversas métricas que giran alrededor de la noción de "complejidad". Los enfoques planteados desde la perspectiva de la actividad cerebral involucrada (en contraste con los patrones del código fuente) se conocen colectivamente como de "complejidad cognitiva" [8]. Para un ejemplo ilustrativo, consúltese [10].

[7] Es decir, corresponde al conjunto de interconexiones existentes en lo que tradicionalmente se denomina "diagrama de flujo".

[8] [20, p598] proporciona un listado (no exhaustivo) conteniendo 38 variantes.

de actualizaciones en línea, etc.[9]. Los puntos de función se utilizan en diversas organizaciones como mecanismo de dimensionamiento de un proyecto de software, puesto que la "funcionalidad" es un concepto relativamente accesible por los usuarios; asimismo se emplea como dato de entrada para diversos métodos de estimación de esfuerzo.

En [20] se estudian ampliamente los puntos de función con el apoyo de información estadística del software norteamericano basada en esta métrica para diversas industrias. Una revisión crítica muy informativa acerca de las motivaciones técnicas, ventajas y desventajas de las principales alternativas a los puntos de función se puede encontrar en [33]. Asimismo, [47] da cuenta del surgimiento de diversas métricas basadas o al menos inspiradas en los puntos de función para la mayoría de nichos importantes.

Desde nuestro punto de vista, la principal utilidad del análisis de puntos de función radica en que obliga a los especialistas a subdividir el problema en unidades más pequeñas y por tanto más comprensibles; en particular, esto puede contribuir significativamente a sustentar la "opinión del experto". Por el lado negativo, el uso de fórmulas con constantes tabuladas representa la usual pretención (usualmente errónea) de diagnosticar casos específicos a partir de promedios históricos carentes de indicadores estadísticos de confiabilidad.

2.4. COCOMO. El Constructive Cost Model en su segunda versión (conocida como COCOMO II) proporciona un modelo matemático/estadístico derivado de información histórica proveniente de 163 proyectos, siendo su primer objetivo el proporcionar estimados correctos y precisos para la planificación de tiempo y los costos de proyectos de software, tanto actuales como a futuro[5].

La estimación de esfuerzo viene dada por la fórmula[10]:

$$PM = A * Size^E * \prod_{i=1}^{n} EM_i,$$

donde PM es el número de meses-persona (el esfuerzo de recursos humanos), A es una constante, $Size$ es el "tamaño" del software (expresado en miles de líneas de código, KSLOC), E es un exponente que modela el impacto de lo que comúnmente se denomina "economía de escala", y EM_i son "multiplicadores de esfuerzo" que corresponden a factores de ajuste que varían en función a diversas características del contexto donde se implementa el desarrollo. Las líneas de código preferentemente se obtienen mediante la aplicación de "puntos de función" y tablas de conversión que aplican un factor dependiendo del lenguaje de programación.

Como vimos, el exponente E representa la capacidad del proyecto de aprovechar el concepto de economía de escala; esto es frecuente en escenarios de trabajo altamente homogéneo, donde la repetitividad de las tareas permite que éstas se completen cada vez más aprisa ya sea por el aprendizaje del ejecutante, como por las herramientas auxiliares de desarrollo que ocasionalmente se crean durante el proyecto[11].

Los factores EM_i son multiplicadores (relativamente cercanos a la unidad) que ajustan el esfuerzo resultante; estos factores son 17 en COCOMO II. A modo de ejemplo, el factor "RELY" (fiabilidad) presenta la siguiente tabla [5, p41]:

Calificación	Muy bajo	Bajo	Nominal	Alto	Muy alto
EM	0.82	0.92	1.0	1.10	1.26

Donde la calificación asociada es:

[9] En el estándar de IFPUG se señalan 14 factores de influencia, cada uno de los cuales se debe expresar en una escala de cero (no aplicable) a 5 (muy influyente.) Notar que esta calificación es esencialmente un producto de la opinión de los expertos. Los puntos de función finalmente se calculan mediante:

$$FP = UFP * [d_1 + (d_2 * TDI)],$$

donde TDI (total degree of influence) corresponde a la suma de las calificaciones de todos los factores de influencia. Las constantes de ajuste d_1 y d_2 son proporcionadas por el estándar.

[10] El ajuste de los parámetros de esta fórmula combinó la opinión de ocho expertos de la industria con la información histórica de los proyectos mencionados, empleando estadística bayesiana[5, p162].

[11] El exponente se obtiene a partir de la expresión:

$$E = 0{,}91 + 0{,}01 * \sum_{j=1}^{5} SF_j,$$

donde SF_j son "factores de escala" experimentalmente tabulados; estos incluyen: similitud con productos anteriores, flexibilidad de la conformidad a especificaciones, cohesión del equipo, etc. Estos factores consiguen que E se establezca cercano y alrededor de la unidad, por lo que $Size^E$ puede resultar mayor o menor que $Size$, según sea el caso.

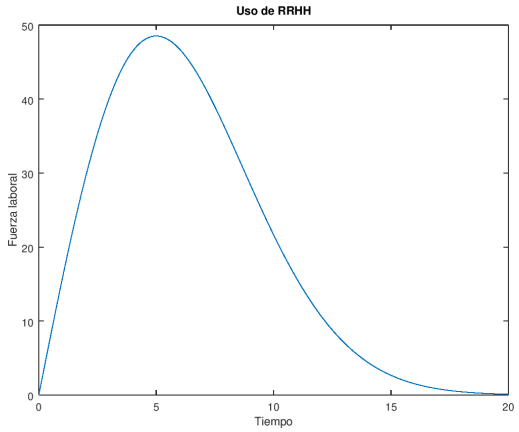

Figura 2.1: Curva de Norden/Rayleigh para fuerza de trabajo.

Descripción	Calificación
Pequeñas Inconveniencias	Muy bajo
Pérdidas Reducidas	Bajo
Pérdidas Moderadas	Nominal
Alta Pérdida Financiera	Alto
Riesgo de Vida Humana	Muy alto

COCOMO (considerando colectivamente sus versiones I y II) es largamente el modelo más discutido en la literatura, y es reportado como el método formal más utilizado [39, p203]. Es empleado en numerosos estudios como referente a ser "superado" en cuanto a su capacidad predictiva. En [6], los autores reflexionan acerca los logros de COCOMO como descendiente de una larga historia de modelos de estimación que lo anteceden.

2.5. Modelo de Putnam/Norden. En base a observaciones experimentales sistemáticas para grandes proyectos de hardware[12], Norden[38] observó que la asignación del personal a lo largo del ciclo de vida de un proyecto complejo, puede modelarse mediante la ecuación:

$$y' = 2Kate^{-at^2},$$

donde y' es la asignación de personal en un determinado momento (cuántas personas están asignadas en el día "t"), K representa el "esfuerzo total" del proyecto, y a es una constante tal que $a = \frac{1}{2t_d^2}$ donde t_d es el momento en el que y' se hace máximo. Un gráfico de $y'(t)$ se presenta en la figura 2.1[13].

La integración de esta función proporciona el esfuerzo acumulado para el proyecto:

$$y = \int_0^t y' = K(1 - e^{-at^2}).$$

Esta magnitud está dada en personas-año u otra unidad análoga. Se observa claramente que $\lim_{t\to\infty} y(t) = K$; es decir, el esfuerzo total del proyecto es K, lo que prueba la anterior indicación que se hizo para este parámetro.

Putnam[42] aplicó este modelo a proyectos de software, extendiendo sus resultados y contribuyendo a su popularización[14]; adicionalmente, Putnam introdujo algunas asunciones complementarias y dedujo la

[12] Según [38, p219] el estudio inicial data del periodo 1956-1964.

[13] Esta ecuación coincide con la función de densidad probabilística de Rayleigh descrita en las ciencias físicas: Si x e y son variables aleatorias con distribución normal y media cero con la misma varianza, entonces $\sqrt{x^2 + y^2}$ sigue la distribución de Rayleigh [40].

[14] La curva de Norden/Rayleigh también fue utilizada en la primera versión de COCOMO [4, p67,92]; sin embargo, el autor observó que su aplicación sólo era conveniente en ciertos tramos del intervalo temporal, situación que dependía de la clase de proyecto considerado. Este enfoque fue abandonado en COCOMO II.

(muy criticada) "ecuación del software"[15].

El modelo de Putnam tuvo un considerable impacto académico y consecuentemente fue criticado por diversos investigadores, algunos de los cuales propusieron sendas alternativas (ver una recopilación de éstas en [51].) Nosotros nos limitaremos a señalar que esta aproximación es simplemente inaplicable para la gran mayoría de los proyectos de software tal como se desarrollan en la actualidad; esto se discute inmediatamente a continuación.

3. Discusión. En esta sección pasamos revista a un conjunto de asunciones y argumentaciones en las que se sostienen diversos modelos de estimación de esfuerzo, tales como los que se han reseñado líneas atrás. En especial, señalamos una serie de dificultades recurrentes de cara a la objetividad de tales modelos, con el consecuente cuestionamiento a la utilidad que aportaría su uso.

3.1. Desarrollo "a lo grande" vs. desarrollo "en pequeño". Al consultar los orígenes de los primeros modelos de estimación de esfuerzo se aprecia rápidamente que sus principales proponentes han pertenecido a grandes organizaciones embarcadas en proyectos de elevada complejidad a cargo de grandes equipos de especialistas. Esto se refleja muy claramente en las curvas de Norden antes reseñadas, las cuales sólo tienen sentido en proyectos con estas características.

Esta visión del desarrollo de software "a lo grande" (donde el computador todavía es un privilegio muy exclusivo), corresponde perfectamente a las décadas 50-70, donde los proyectos en cuestión son principalmente de índole científico/militar y/o gubernamental[16]. Es una era de especialistas atendiendo a otros especialistas, empleando un conjunto aún muy reducido de tecnologías.

Con la llegada del microcomputador (décadas 80-90) podemos encontrar especialistas atendiendo a un numeroso grupo de no-especialistas. El software automatiza diversos procesos de organizaciones de todo tamaño y especie, y se convierte en herramienta de trabajo para toda clase de profesionales y entusiastas.

Este proceso de masificación del software impulsa el desarrollo de múltiples emprendimientos a pequeña escala, para lo cual se conforman equipos de desarrollo dedicados a nichos cada vez más restringidos a fin de satisfacer las necesidades más especializadas de los clientes. En consecuencia, estos equipos son ahora de tamaño muy reducido y deben ejecutar los desarrollos en plazos cada vez más ajustados: sus integrantes son normalmente los mismos desde el principio hasta el final del proyecto puesto que no hay tiempo para reclutamientos sobre la marcha; asimismo, es bien sabido que el personal que se incorpora a un proyecto ya en curso, normalmente lo retrasa [7]. A esta forma de enfrentar los proyectos la denominamos desarrollo "en pequeño".

Esto quiere decir que la dinámica de los proyectos en la actualidad es radicalmente distinta a la que se daba en los albores de la ingeniería de software; naturalmente, también existen proyectos de gran envergadura en ciertos nichos; sin embargo, en términos relativos el desarrollo de proyectos "en pequeño" representa largamente la mayor proporción en esta actividad: en este nuevo escenario de pequeños equipos (de tamaño esencialmente invariante) operando con plazos muy breves, la antigua curva de Norden se convierte en una función constante de reducida magnitud y duración. Esta idea se ilustra en la figura 3.1.

Lo anterior implica que los modelos de estimación tradicionales que describen la variación temporal de un personal numeroso, son sencillamente inaplicables para los proyectos que usualmente acometen los desarrolladores de hoy.

3.2. Un mundo ágil: la variabilidad de los requerimientos. El esfuerzo requerido en un proyecto depende directamente de las necesidades (requerimientos) a satisfacer; lógicamente, un conjunto distinto de requerimientos involucrará un esfuerzo distinto. Sin embargo, es el caso que los requerimientos de cualquier proyecto de software no trivial en la actualidad *normalmente* varían en forma considerable durente el desarrollo, por lo que resultan distintos a los que se consideraron inicialmente; ergo, el estimado de esfuerzo inicial *normalmente* es impreciso y/o inútil.

En este orden de ideas, una fuerte variación en los requerimientos tiene como correlato una fuerte variación en el esfuerzo involucrado en el desarrollo. En ese sentido, es menester indagar acerca del grado de variabilidad de los requerimientos en los proyectos: si nos atenemos a las tendencias más populares en el desarrollo de software, debemos concluir que esta variabilidad sólo ha ido en aumento a través de los años.

[15] La "ecuación del software" viene dada por:

$$S_s = C_k K^{1/3} t_d^{4/3},$$

donde S_s es el "tamaño" del software desarrollado en el proyecto, y C_k es una constante que representa el "estado de la tecnología" del momento. En teoría, esta ecuación permite efectuar una planificación óptima de los recursos para un proyecto; sin embargo, su aplicación genera resultados paradójicos, y ha sido ampliamente criticada por falta de rigor en su deducción y verificación experimental [29, 48].

[16] Poco tiempo después se incluye el soporte a la gestión comercial de grandes empresas y la banca. Una detallada recopilación histórica para la ingeniería de software subdividida en décadas puede consultarse en [21].

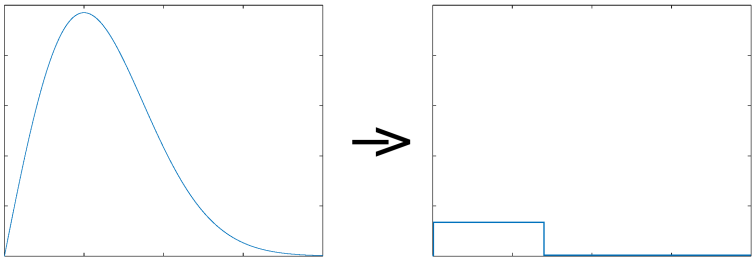

Figura 3.1: Proyectos “a lo grande” vs. desarrollo “en pequeño”: Fuerza laboral a través del tiempo.

En el plano metodológico, con el cambio de milenio se han popularizado ampliamente las denominadas “metodologías ágiles” donde el énfasis gira en torno a la plena satisfacción del cliente, para lo cual sus requerimientos son re-elaborados iterativamente puesto que su conocimiento a priori se asume imposible. Luego, la continua variación en los requerimientos se acepta como inexorable e incluso benéfica. Como ilustración, uno de los principios del ya clásico “Manifiesto ágil”[3] reza:

“Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.”

En el frente tecnológico, la ubicuidad del Internet ha elevado la representatividad numérica de los desarrollos de software orientados a la web. Al respecto [13, p102] reflexiona irónicamente acerca de los proyectos web:

- Libres de requerimientos (los clientes aún no saben lo que quieren realmente)
- Desarrollo rápido (los clientes pueden no saber lo que quieren, ¡pero saben que lo quieren pronto!)
- Poca duración y pequeñez (hablan de desarrollo “3x3” - tres personas por tres meses.)

3.3. Los “Datos Históricos” que justifican los Modelos. Para la sustentación de los modelos de estimación de esfuerzo, muchos investigadores han recurrido a bases de datos históricas públicas (“datasets” en adelante), que consolidan diversas métricas de proyectos ya ejecutados, pero cuya idoneidad deja mucho que desear por diversos motivos.

El primer inconveniente corresponde a la ausencia de información necesaria: los datos deben provenir de proyectos reales y contener precisamente la información requerida por los modelos a ser contrastados; evidentemente, esto no siempre es posible, pues las costumbres de recolección de métricas de proyectos varían de organización en organización. Esto significa que diversos modelos nunca alcanzan a ser puestos en práctica con información estadísticamente significativa, por la simple razón que los datos necesarios no están disponibles. Según [20, p193] la mayoría de empresas (de desarrollo de software) simplemente no registra datos, salvo un pequeño subconjunto de las actividades realizadas.

En segundo lugar, la calidad de los datos es cuestionable. En las últimas dos décadas se han acumulado diversas bases de datos de proyectos, notablemente las que actualmente están organizadas en el ISBSG [17], así como las documentadas en [18]. Los datos recolectados en estos repositorios son producto de la “donación” de diversas organizaciones, las cuales no tienen mayor incentivo para asegurar su veracidad y/o exactitud. De acuerdo con [20, p192] es un hecho lamentable que la mayoría de sistemas de seguimiento corporativo para los costos y el esfuerzo (dólares, horas de trabajo, meses-persona, etc.) son incorrectos y logran omitir desde 30 a más de 70 por ciento del esfuerzo real aplicado a los proyectos de software.

En esta línea [15, p172] da cuenta de la inutilidad de la gran mayoría de mediciones y estimaciones publicadas por la industria del software: se recolecta grandes cantidades de información acerca de prácticas que están pobremente descritas o son defectuosas desde el principio. Esa información es entonces diseminada en forma tal que hace casi imposible de confirmar o validar su significancia.

De otro lado, [22, p221] nos advierte que los datos publicados están representados en unidades incompatibles: la industria del software es única en tener más variantes métricas que cualquier otra disciplina de ingeniería en la historia, combinada con una casi total ausencia de reglas de conversión de una métrica a otra.

Siguiendo a este mismo autor, la dimensión temporal tampoco está exenta de crítica: La industria de software [al año 2016] no tiene ningún método estándar para medir ya sea la fecha de inicio o de término de los proyectos de software mayores. Como resultado, la información histórica de su “programación cronológica” es altamente sospechosa[22, p158].

Otro efecto negativo se advierte cuando se tiene en consideración que un porcentaje considerable de los proyectos son cancelados antes de su conclusión[17], lo que ocurre usualmente cuando el esfuerzo dedicado

[17]Existen diversos estudios (normalmente basados en encuestas) acerca de la tasa de fallo de los proyectos de software. Por

(y/o el tiempo transcurrido) sobrepasa algún umbral de tipo comercial o contractual. En tal escenario el proyecto usualmente no es reportado, y su ausencia implica que los modelos poseen un fuerte sesgo hacia aquellos proyectos que (por cualesquiera razones) sí lograron completarse.

Somos de la opinión que esta situación se deriva especialmente de la dificultad y/o reticencia que tienen los administradores y sus organizaciones para admitir sus problemas de gestión de proyectos. Estos problemas rara vez son generados por incompetencia o falta de interés, sino por el contexto de alta incertidumbre usualmente asociado a los proyectos de software: requerimientos vagamente definidos y prioridades cambiantes; interrupciones para atender emergencias operativas, rotación del personal con conocimiento privilegiado, sobretiempos de dudosa legalidad, tiempo "robado" para la investigación de nuevas herramientas de desarrollo, etc. Todos estos factores son inevitables (si bien pueden ser mitigables), pero tienen en común que exponen a los especialistas a la crítica, así como a demandas legales contra sus organizaciones.

En conclusión, los datos históricos que suelen emplearse para la justificación empírica de los modelos son incompletos, imprecisos, incompatibles y sesgados. Sin embargo, esto no ha sido obstáculo para que sigan siendo empleados en la calibración de los modelos tradicionales y experimentales, así como en sendas contrastaciones acerca de su potencial efectividad.

3.4. La precisión de los modelos en relación al tiempo. En todas las disciplinas que poseen alguna clase de modelo predictivo, el nivel de precisión numérica asociado a sus resultados (o predicciones) constituye un elemento imprescindible para su aplicación efectiva. Sin embargo, en los modelos de estimación de esfuerzo de software esto se ignora o se considera muy a la ligera. Por ejemplo, [35, p39,253] recomienda calcular los rangos de incertidumbre de los estimados a partir del valor puntual obtenido por el modelo, aplicando un conjunto de factores que son extraídos de la literatura de COCOMO; estos rangos son dinámicos, en el sentido que la estimación se hace cada vez más precisa conforme discurre la ejecución del proyecto[18].

Sin embargo, el problema radica en el pobre nivel de precisión de tales predicciones consideradas "exitosas". Concretamente, para una estimación de esfuerzo "X" obtenida al inicio del proyecto (cuando tal estimación tiene mayor valor), los autores mencionados reportan una (muy elevada) incertidumbre correspondiente al rango $[25\,\%X - 400\,\%X]$. Este intervalo se reduciría paulatinamente conforme se avanza con el proceso de análisis, concluyendo en un excelente $[90\,\%X - 110\,\%X]$ al completarse las tareas denominadas de "diseño detallado".

Lamentablemente, en la práctica es menester estimar y presupuestar los proyectos antes que éstos sean iniciados[19], justamente cuando los modelos proporcionan respuestas con excesivo margen de error y -por tanto- de poca valía.

Este problema se exacerba para los proyectos ejecutados con metodologías no tradicionales en las que el diseño evoluciona en paralelo con la construcción iterativa del código fuente: la estimación de esfuerzo (y el presupuesto asociado) también estaría sujeta a evolucionar.

3.5. La Objetividad de los Datos de Entrada. Diversos modelos de estimación utilizan como punto de partida (o dato de entrada principal) al "tamaño total" del código fuente de los programas a ser creados en el proyecto en cuestión: este tamaño es considerado el principal determinante para el esfuerzo asociado.

El tamaño del código fuente se debe obtiener a partir de los requerimientos del proyecto, usualmente mediante el análisis de puntos de función u otros métodos análogos, en donde normalmente interviene la opinión del experto.

Si estos modelos de estimación en realidad no parten de los requerimientos del proyecto, sino de un "tamaño estimado" (más o menos subjetivo) proporcionado por los especialistas, entonces sus resultados necesariamente acarrean esta carga subjetiva: su exactitud estará siempre supeditada a la calidad de la opinión del experto. Lamentablemente, un modelo con resultados de exactitud desconocida tiene una utilidad muy reducida.

Este problema no sólo se manifiesta con el estimado del tamaño total, sino con la medición y/o calificación de los datos de entrada complementarios; para ilustrar este punto, veamos un ejemplo tomado de COCOMO II [5, p34]. En este modelo la "cohesión del equipo" es un dato de entrada (denominado "factor") que se define así:

El factor escala de cohesión del equipo da cuenta de las fuentes de turbulencia y entropía del proyecto debido a dificultades en sincronizar a los actores relevantes del proyecto.

---

ejemplo, el estudio [12] realizado en 2005 y 2007 intenta superar algunas contradicciones y ausencias frecuentes en las publicaciones acerca de este tópico. En sus conclusiones señala: La tasa de cancelación de proyectos de TI estuvo en el rango de 11.5 a 15.5 %. [...] La tasa combinada de proyectos cancelados pero no exitosos estuvo entre 26 % y 34 %.

[18] El gráfico usual para esta idea se conoce como el "cono de incertidumbre"; éste puede consultarse en las referencias mencionadas en esta sección. Por su parte, la referencia original de COCOMO [4, p310] al describir tales factores sólo menciona (en un pie de página) que tales rangos han sido determinados *subjetivamente*, y representan límites de confianza al 80 %. Siguiendo la práctica estadística usual, esto se debe interpretar como modelos cuyo "éxito" predictivo ocurre en aproximadamente el 80 % de ocasiones.

[19] Usualmente dependerá del presupuesto si el proyecto efectivamente se inicia.

El modelo prescribe el uso de una tabla de escala Likert (análoga a la presentada en la sección 2.4) que debe ser empleada por el usuario del modelo para *calificar* al factor. Por tanto, el especialista deberá ser capaz de predecir si el equipo de trabajo (que podría incluir personal recién reclutado) será -a lo largo del proyecto- muy cohesionado, algo cohesionado, neutralmente cohesionado, poco cohesionado, o muy poco cohesionado.

Esta forma de calificación o "medición" resulta ser de naturaleza totalmente subjetiva. En la práctica, esta tarea será encargada a un experto responsable del proyecto, con lo cual -tras un gran rodeo- se termina reintroduciendo por la puerta falsa la opinión subjetiva de los especialistas.

Todo lo anterior nos conduce a proponer una obvia recomendación de cara a nuevos modelos de estimación: en lo posible, éstos deberían ser alimentados *directamente* por los requerimientos del proyecto. Sin duda esto no es sencillo, puesto que los requerimientos usualmente son especificados de un modo extremadamente informal, y con mucha frecuencia provienen de actores no especializados.

En este orden de ideas, resulta razonable evaluar la introducción de una especificación intermedia formalizada para los requerimientos[20] como etapa previa a la aplicación de los modelos, a fin de proporcionarles datos de entrada objetivos. Notar que el uso de especificaciones formalizadas incrementaría las oportunidades de automatización en las futuras herramientas de estimación.

3.6. La Contrastación de los Modelos. Los investigadores interesados en la estimación del esfuerzo del software han dedicado mucho esfuerzo a la comparación empírica de los modelos a fin de dilucidar cuáles son los más convenientes y/o recomendables. Estas comparaciones requieren (al menos) de un amplio conjunto de datos históricos de alta calidad, así como de algunas métricas objetivas de contrastación a ser aplicadas sobre los resultados estimados. Como ya vimos anteriormente, la industria no ha logrado generar los datos históricos necesarios; sin embargo, las métricas utilizadas para procesarlos tampoco han escapado a la crítica.

Siguiendo a [26], el problema fundamental con estas métricas de comparación de modelos se deriva de la inexistencia de un acuerdo en lo que significa que un estimado sea mejor que otro. Concretamente, la métrica más utilizada en la literatura para comparar la capacidad predictiva de los modelos es la "media del error total relativo" (MMRE)[21], y como en toda métrica basada en el error, un valor más reducido se interpreta como una mayor exactitud (alcazada por el modelo); sin embargo, en la práctica esto resulta ser falso en muchos escenarios[22]. En esta línea, [31, p81] proporciona un análisis acerca de las propiedades estadísticas del MMRE, señalando que esta métrica no es efectiva para evaluar la exactitud de las predicciones; por el contrario, correspondería a una métrica de dispersión de la variable aleatoria dada por $\frac{\widehat{x_i}}{x_i}$.

Estas métricas de dudosa fundamentación, al ser aplicadas sobre datos aún más cuestionables, explican una larga cantidad de estudios no concluyentes y/o mutuamente contradictorios acerca de las supuestas bondades predictivas de ciertos modelos en relación a otros, lo que sin duda menoscaba la reputación de este campo de estudio en su conjunto.

4. Conclusiones. A partir de una selección representativa de modelos de estimación de esfuerzo se ha planteado un conjunto de ideas que pueden ser de utilidad (o al menos, servir de advertencia) para futuras investigaciones. En particular, las nuevas propuestas de modelamiento deben evitar las usuales comparaciones estériles sustentadas en una larga acumulación histórica de datos incontrastables, cuando no espurios.

Las (nuevas) bases de datos históricas acerca de proyectos de desarrollo de software no pueden ser construidas a partir de los reportes usuales de las organizaciones, al ser éstos esencialmente tendenciosos, de contenido heterogéneo, y construidos en base a términos sin definición explícita. En el presente estadio

[20] Existen diversos lenguajes formales de modelamiento de requerimientos [27, p33], los cuales son un punto de partida natural para lo que proponemos. Estos lenguajes capturan totalmente el problema que intenta resolver el proyecto, permitiendo la aplicación de métodos formales orientados a garantizar la corrección al momento de la ejecución de los programas. Lamentablemente su uso conlleva una elevada complejidad y considerable esfuerzo, lo que es prohibitivo en la mayoría de escenarios. En ese sentido, futuras investigaciones deben explorar alternativas simplificadas y orientadas exclusivamente a la estimación.

[21] Denotando por x_i al esfuerzo invertido en un proyecto, si la estimación por algún modelo modelo fue $\widehat{x_i}$, entonces se define la magnitud del error relativo (Magnitude of Relative Error, MRE) como:

$$MRE = |\frac{\widehat{x_i} - x_i}{x_i}|.$$

A su turno, el MMRE (Mean Magnitude of Relative Error o Media del error total relativo) para un conjunto de estimaciones (aplicación del modelo a múltiples proyectos) se define mediante:

$$MMRE = \frac{1}{n}\sum_{i=1}^{n} |\frac{\widehat{x_i} - x_i}{x_i}|.$$

Para otras métricas relacionadas, consultar por ejemplo [44] y [43].

[22] En [26, p4] se ilustra este punto con un ejemplo: un estimado de esfuerzo para el cual el esfuerzo real es 300 % del resultado de la estimación (MRE=0.67) resulta ser *más exacto* que un estimado de esfuerzo donde el esfuerzo real es 59 % del estimado (MRE=0.69.)

de la industria del software, consideramos que la única fuente de información objetiva la constituyen los repositorios de código fuente, especialmente los sistemas de control de versiones que describen su evolución detallada a través del tiempo.

De otro lado, demostramos que los modelos de estimación usualmente introducen implícitamente el juicio subjetivo del especialista en lo que respecta al dimensionamiento (tamaño total) del proyecto. Como alternativa de investigación se propuso el empleo de especificaciones de requerimientos formalizadas.

En cuanto a la necesidad práctica de realizar estimaciones con los métodos hoy disponibles, rescatamos dos aproximaciones: en primer lugar, la recomendación de consolidar resultados provenientes de diversos métodos: [35, p165] señala que ninguna técnica individual de estimación es perfecta, así que el uso de múltiples aproximaciones es útil en muchos contextos. Los productores [de estimados] comerciales más sofisticados usan al menos tres diferentes aproximaciones de estimación y luego buscan la convergencia o dispersión entre los estimados.

En segundo lugar, toda la discusión precedente nos conduce a desconfiar de los modelos tradicionales que intentan abordar el problema en su máxima generalidad; por el contrario, debemos esperar resultados más satisfactorios con métodos y/o modelos orientados a estimaciones en ámbitos más restringidos[23]. Asimismo, si aceptamos que no todo proyecto es susceptible de una estimación objetiva, entonces es razonable reorientar la investigación hacia las condiciones y/o requisitos que sí la hacen posible.

ORCID and License
Diego Bravo-Estrada https://orcid.org/0000-0002-7439-0772
Roxana López-Cruz https://orcid.org/0000-0002-7703-5784

Referencias

[1] Albrecht AJ. Measuring application development productivity. InProc. joint share, guide, and ibm application development symposium 1979 (pp. 83-92).
[2] Arnuphaptrairong T. The state of practice of software cost estimation: Evidence from thai software firms. InProceedings of the International MultiConference of Engineers and Computer Scientists 2018 (Vol. 2).
[3] Beck K, Beedle M, Van Bennekum A, Cockburn A, Cunningham W, Fowler M, Grenning J, Highsmith J, Hunt A, Jeffries R, Kern J. Manifesto for agile software development. https://agilemanifesto.org/, 2001
[4] Boehm BW. Software engineering economics. IEEE transactions on Software Engineering. 1984 Jan(1):4-21.
[5] Barry W. Boehm, Clark, Horowitz, Brown, Reifer, Chulani, Ray Madachy, and Bert Steece. 2000. Software Cost Estimation with Cocomo II with Cdrom (1st. ed.). Prentice Hall PTR, USA.
[6] Boehm BW, Valerdi R. Achievements and challenges in cocomo-based software resource estimation. IEEE software. 2008 Aug 19;25(5):74-83.
[7] Brooks Jr FP. The mythical man-month: essays on software engineering. Pearson Education; 1995 Aug 2.
[8] Cant SN, Jeffery DR, Henderson-Sellers B. A conceptual model of cognitive complexity of elements of the programming process. Information and Software Technology. 1995 Jan 1;37(7):351-62.
[9] Chemuturi M. Software estimation best practices, tools & techniques: A complete guide for software project estimators. J. Ross Publishing; 2009 Jul 15.
[10] Chhabra JK, Aggarwal KK, Singh Y. Code and data spatial complexity: two important software understandability measures. Information and software Technology. 2003 Jun 1;45(8):539-46.
[11] Coulter NS. Software science and cognitive psychology. IEEE Transactions on Software Engineering. 1983 Mar(2):166-71.
[12] El Emam K, Koru AG. A replicated survey of IT software project failures. IEEE software. 2008 Aug 19;25(5):84-90.
[13] Glass RL. Some heresy regarding software engineering. IEEE Software. 2004 Jul 6;21(4):104-3.
[14] Halstead MH. Elements of Software Science (Operating and programming systems series). Elsevier Science Inc.; 1977 May 1.
[15] Hetzel WC. The sorry state of software practice measurement and evaluation. Journal of Systems and Software. 1995 Nov 1;31(2):171-9.
[16] IFPUG. International Function Point User's Group. Function Point Counting Practices Manual, 2010.
[17] ISBSG. International software benchmarking standards group. http://isbsg.org/, n.d. Accedido el 13/01/2019.
[18] C. Jones. Sources of Software Benchmarks. 2011. https://insights.cermacademy.com/14-sources-of-software-benchmarks-c-capers-jones-2/ .
[19] Jones C. Software project management practices: Failure versus success. CrossTalk: The Journal of Defense Software Engineering. 2004 Oct;17(10):5-9.
[20] Jones C. Applied software measurement. McGraw-Hill Education; 2008.
[21] Jones C. The technical and social history of software engineering. Addison-Wesley; 2013 Nov 21.
[22] Jones C. A guide to selecting software measures and metrics. CRC Press; 2017 Mar 3.
[23] Derek Jones. Software effort estimation is mostly fake research. http://shape-of-code.coding-guidelines.com/2021/01/17/software-effort-estimation-is-mostly-fake-research/, 2021.
[24] Bertrand Meyer. The origin of software engineering. https://bertrandmeyer.com/2013/04/04/the-origin-of-software-engineering/, 2013.
[25] Jørgensen M. A review of studies on expert estimation of software development effort. Journal of Systems and Software. 2004 Feb 1;70(1-2):37-60.

[23] A modo de ejemplo, el modelo de [30] nos demuestra que el problema de estimación en realidad "ya está resuelto", siempre que se circunscriba a estimar proyectos similares a otros ya concluidos.

[26] Jørgensen M. A critique of how we measure and interpret the accuracy of software development effort estimation. In First International Workshop on Software Productivity Analysis and Cost Estimation. Information Processing Society of Japan, Nagoya 2007 Dec 4.
[27] Jureta I. The Design of Requirements Modelling Languages. Springer International Publishing; 2015.
[28] Keshta IM. Software cost estimation approaches: A survey. Journal of Software Engineering and Applications. 2017 Sep 28;10(10):824.
[29] Kitchenham B, Taylor NR. Software cost models. ICL technical journal. 1984 May;4(1):73-102.
[30] Kitchenham B, Linkman S. Estimates, uncertainty, and risk. IEEE Software. 1997 May;14(3):69-74.
[31] Kitchenham BA, Pickard LM, MacDonell SG, Shepperd MJ. What accuracy statistics really measure. IEE Proceedings-Software. 2001 Jun 1;148(3):81-5.
[32] Laird LM, Brennan MC. Software measurement and estimation: a practical approach. John Wiley & Sons; 2006 Jun 5.
[33] Lokan CJ. Function points. Advances in Computers. 2005 Jan 1;65:297-347.
[34] McCabe TJ. A complexity measure. IEEE Transactions on software Engineering. 1976 Dec(4):308-20.
[35] McConnell S. Software estimation: demystifying the black art. Microsoft press; 2006 Feb 22.
[36] Menzies T, Yang Y, Mathew G, Boehm B, Hihn J. Negative results for software effort estimation. Empirical Software Engineering. 2017 Oct;22:2658-83.
[37] Molokken K, Jorgensen M. A review of software surveys on software effort estimation. In2003 International Symposium on Empirical Software Engineering, 2003. ISESE 2003. Proceedings. 2003 Sep 30 (pp. 223-230). IEEE.
[38] Norden PV. Project life cycle modeling: Background and application of the life cycle curves. US Army Computer Systems Command. 1977 Aug 21.
[39] Osmanbegović E, Suljić M, Agić H. A review of estimation of software products development costs. Ekonomski Vjesnik/Econviews-Review of Contemporary Business, Entrepreneurship and Economic Issues. 2017 Jun 29;30(1).
[40] Papoulis A, Unnikrishna Pillai S. Probability, random variables and stochastic processes. 2002.
[41] Pmi I. Software Extension to the PMBoK Guide. Project Management Institute, Philadelphia, USA. 2013.
[42] Putnam LH. A general empirical solution to the macro software sizing and estimating problem. IEEE transactions on Software Engineering. 1978 Jul(4):345-61.
[43] Sharma K, Garg R, Nagpal CK, Garg RK. Selection of optimal software reliability growth models using a distance based approach. IEEE Transactions on Reliability. 2010 May 6;59(2):266-76.
[44] Shepperd M, Cartwright M, Kadoda G. On building prediction systems for software engineers. Empirical Software Engineering. 2000 Nov;5:175-82.
[45] Shepperd M, Ince DC. A critique of three metrics. Journal of systems and software. 1994 Sep 1;26(3):197-210.
[46] Stepanek G. Software project secrets. George Stepanek; 2005.
[47] Stutzke RD. Estimating software-intensive systems: projects, products, and processes. Pearson Education; 2005 Apr 26.
[48] Suelmann H. Putnam's effort-duration trade-off law: is the software estimation problem really solved?. In 2014 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement 2014 Oct 6 (pp. 79-84). IEEE.
[49] Trendowicz A, Jeffery R, Trendowicz A, Jeffery R. Classification of Effort Estimation Methods. Software Project Effort Estimation: Foundations and Best Practice Guidelines for Success. 2014:155-208.
[50] Vera T, Ochoa SF, Perovich D. Survey of software development effort estimation taxonomies. Computer Science Department, University of Chile: Santiago, Chile. 2017 Dec.
[51] Verhoef C. Quantitative IT portfolio management. Science of computer programming. 2002 Oct 1;45(1):1-96.
[52] Wen J, Li S, Lin Z, Hu Y, Huang C. Systematic literature review of machine learning based software development effort estimation models. Information and Software Technology. 2012 Jan 1;54(1):41-59.